Learning to Walk - Reinforcement Learning on FrozenLake

Hannah Li and Teng Zhang

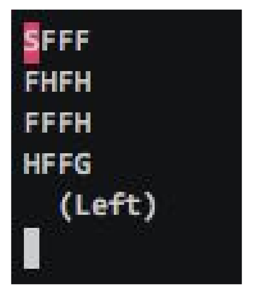
Department of Management Science & Engineering, Stanford University

Open AI Gym

Open AI Gym Environment

- Toolkit for developing and comparing reinforcement learning algorithms in games such as Ms. Pacman, Roulette, and Asteroids
- Formulated as episodic MDP reinforcement learning problems

FrozenLake-v0



- Agent walks on a frozen lake attempting to retrieve a lost frisbee
- Lake is slippery - transitions are stochastic with respect to state and action
- Game ends when agent finds frisbee ('G') or falls into a hole ('H')
- Reward 1 for finding frisbee, 0 otherwise

Algorithms

Model-based Algorithm

- PSRL
  - Initialize prior distribution
  - For every episode:
    - update posterior distribution
    - sample MDP from posterior
    - generate and apply corresponding optimal policy

Value function learning

- General framework
  - live($\mathbf{\Theta}$,$\mathbf{Q}$,cache,act,learn)
    - $\mathbf{\Theta}$: value function parameter set
    - $\mathbf{Q}$: value function family
    - cache: function to maintain the buffer of observations
    - act: function to generate actions according to estimation
    - learn: function to update the parameter $\theta \in \mathbf{\Theta}$

- Learn_LSVI
  - Initialize θ
  - Planning horizon H
  - for h in (1,...,H):
    - regress: *min* [square error sum] + [penalty function]

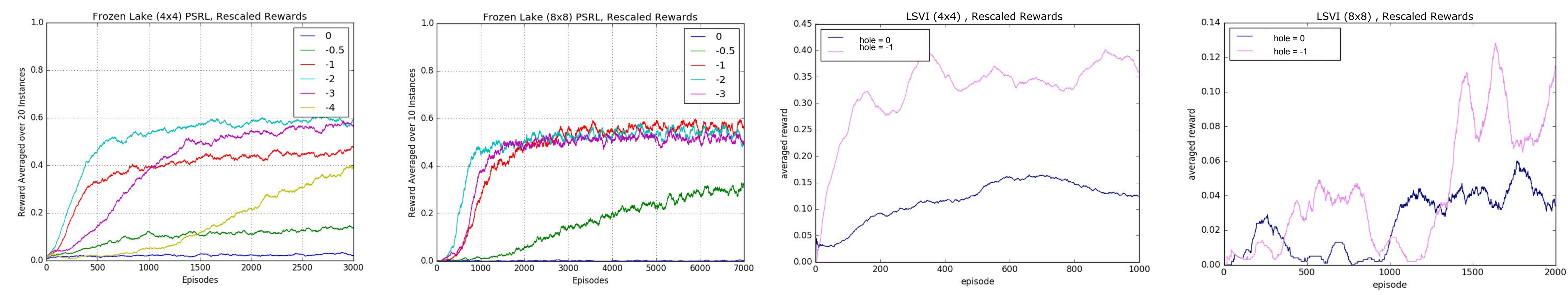
- Learn_LSVI_TD
  - set θ to be the last learned θ
  - N: number of batches
  - for n in (1,...,N):
    - sample batch
    - calculate gradient of loss function
    - do TD update

      θ = θ - learning_rate*gradient

Results

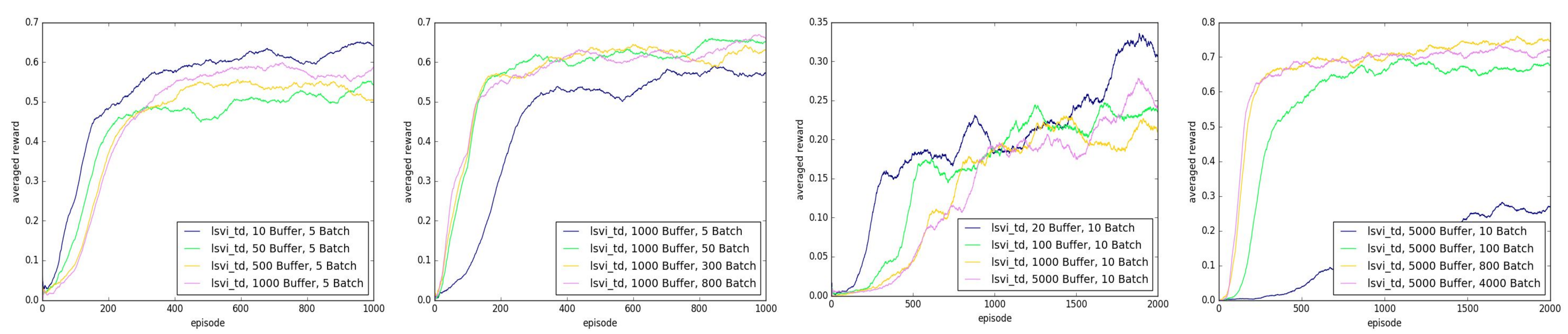
Augmenting rewards

We augment the reward associated with falling into a hole 'H' in the learning algorithm, while keeping the reward 1 for finding the goal 'G' and 0 otherwise. We judge the performance of the algorithm using the original rewards (where we get +1 for getting to 'G' and 0 otherwise). Tuning the augmented rewards greatly increases the performance of the algorithm.



LSVI_TD buffer size and number of batch

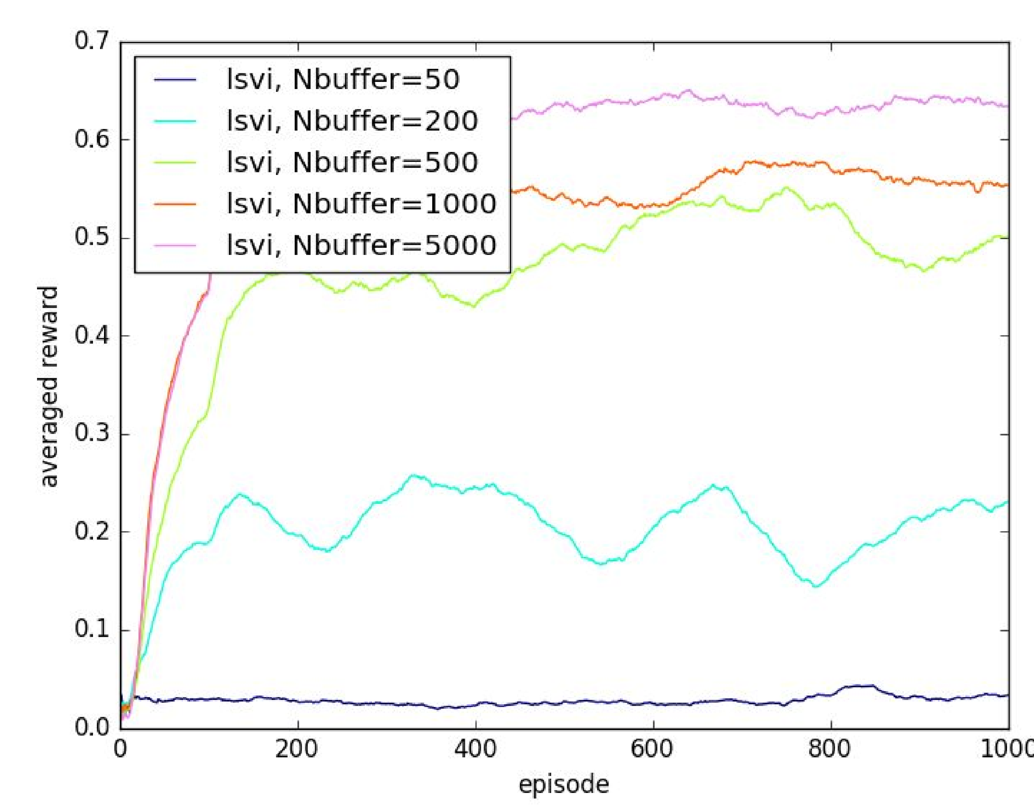
We ran experiments on buffer sizes and number of batches for LSVI_TD. We examine the results for a fixed number of batches and varying buffer sizes as well as a fixed buffer size and varying number of batches.



Efficiency of LSVI and LSVI_TD

The result shows that LSVI_TD is not only computationally cheaper, but also more information efficient in the learning performance.

To achieve comparative performance, LSVI needs a sample size of around 1000, while LSVI_TD only needs a buffer of size 10 and 5 batches.



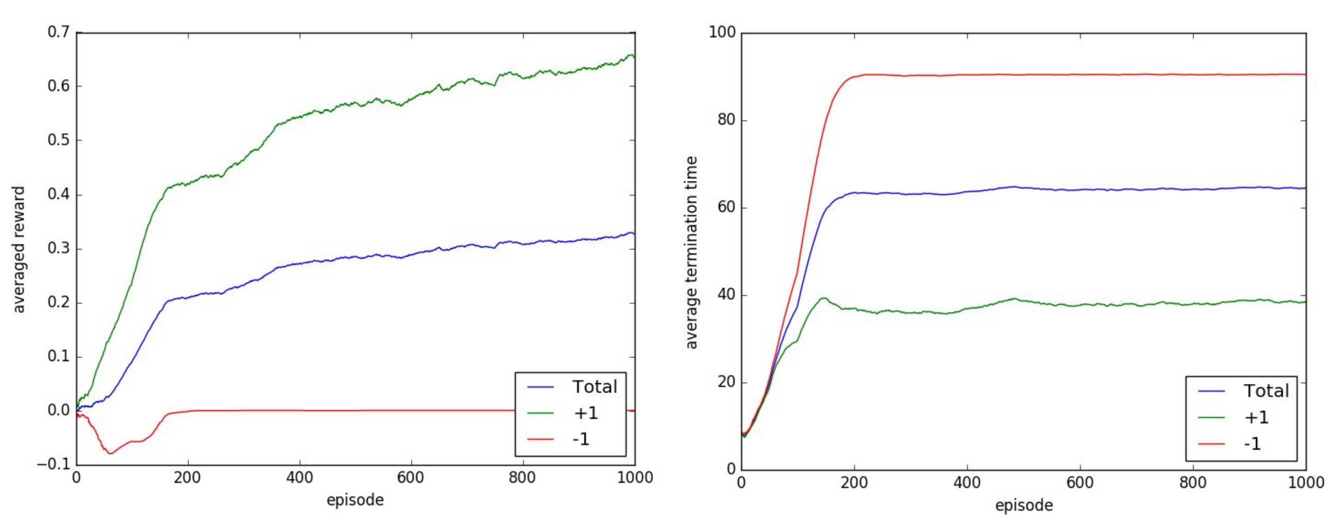
Discussion

Efficiency of LSVI and LSVI_TD

- The computation of lsvi_td is easier, since it does only a gradient update in each step, while LSVI_TD needs to solve a least-square problem every time.
- LSVI does not use the learned parameters to do updates, which is inefficient compared to LSVI_TD which uses the current estimated parameter as the starting point.

Discussion of Prior

- In the experiment, we are actually using a misspecified prior, where the agent does not know where the goal is or if the goal will have positive or negative reward
- To tackle this problem, consider a revised version of FrozenLake, where the reward for reaching the goal is -1 or 1 with equal probability.
- The simulation result shows that LSVI_TD can also learn well under this setting, where the prior we are using is unbiased.



Other parameters

- We also do experiments on other parameters concerned in these algorithms, for example, the learning rate, the epsilon in epsilon-greedy_act, etc.
- The monotonicity shown in some of the experimental results can shed some light on parameter tuning for practical purposes.

Future DIrections

Speed of getting reward

- Our current algorithm has no incentive to reach the goal quickly. We can modify our algorithm to learn to reach the goal in fewer steps.

Sampling method

- There could be clever way of sampling batch from buffer other than random

Deep exploration

- We can experiment with methods to encourage the algorithm to take actions that will lead to future exploration.